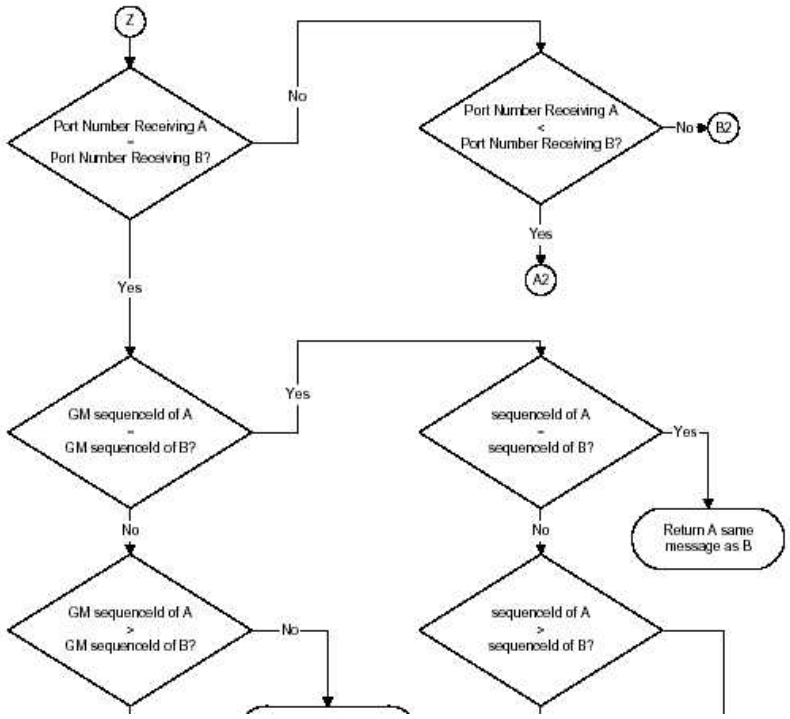
(Last updated: June 7, 2004)

This document contains questions and responses on various aspects of IEEE-1588. These questions are not requests for an interpretation of the meaning of the standard or the resolution of apparent error. As such they are not 'interpretations' as described by the IEEE standards documents and represent only the opinions of the responders.

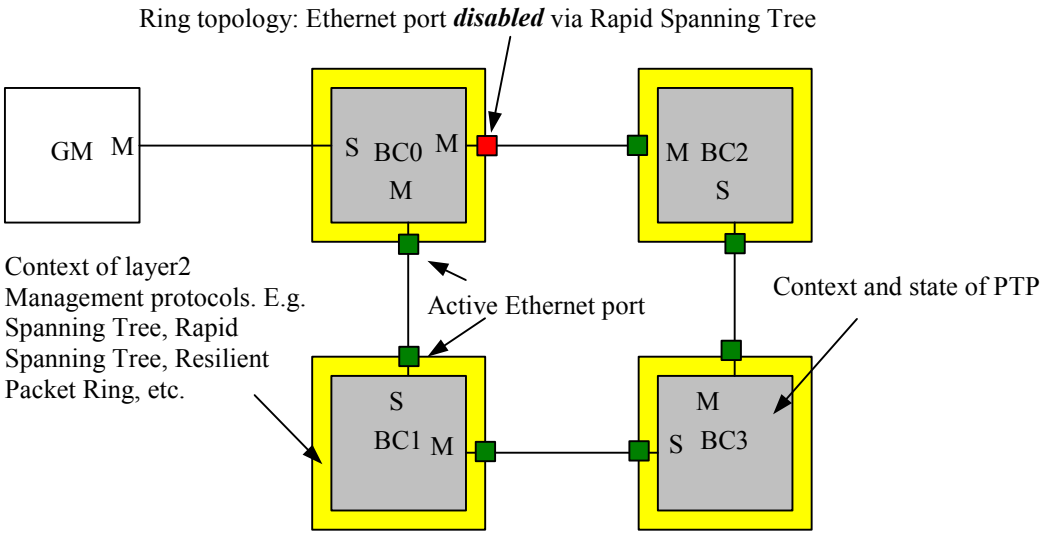
Submitted questions are in BLACK. Responses are in RED.

QUESTION 1: Possible optimization based on layer 2 protocols (submitted March 19, 2003)
Why does the BMC algorithm decides on the PortNumberReceiving before considering the GM SequenceID ? (please see the following figure source Figure 18)

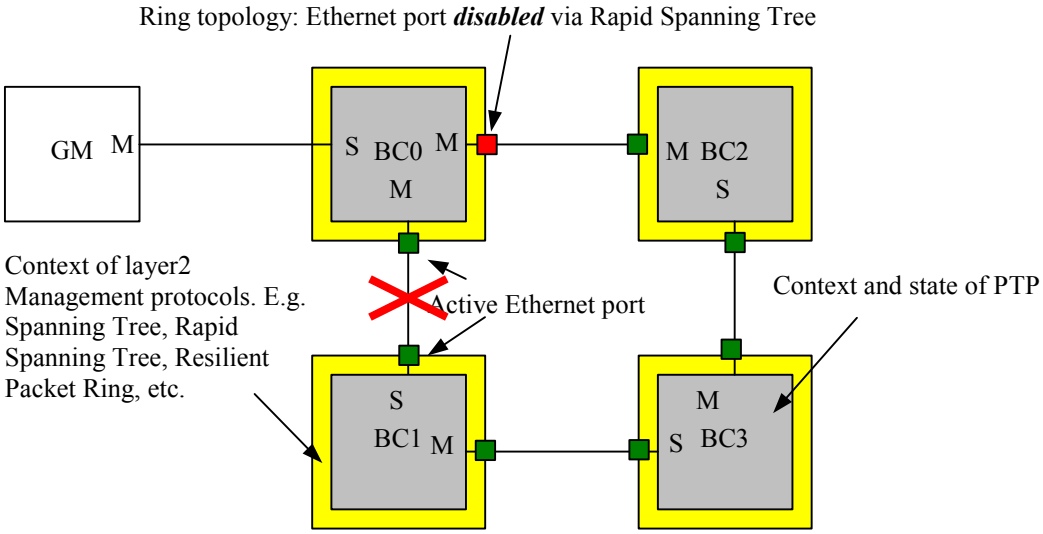


Consider The following example.

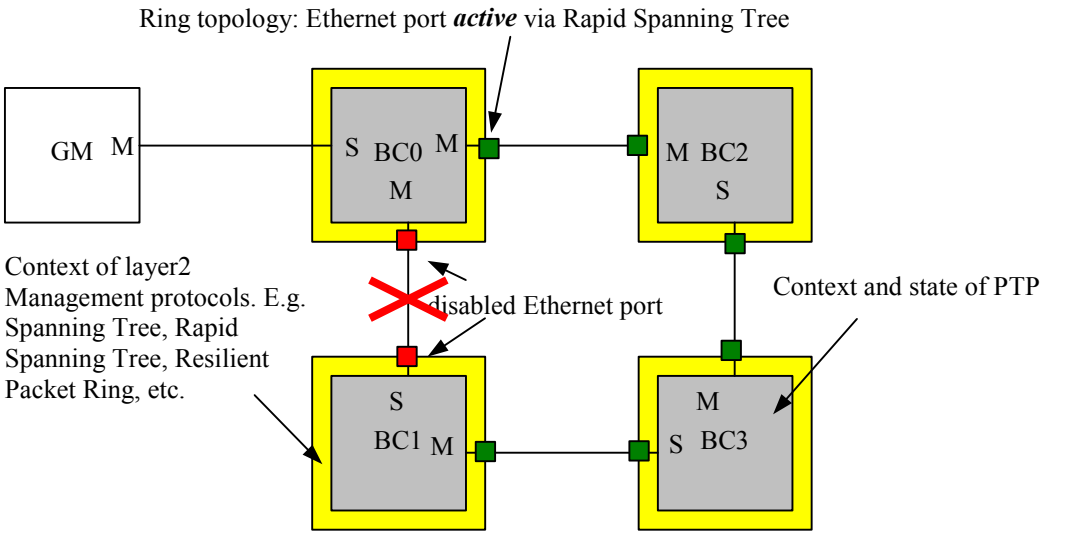
The following figure illustrates the initial configuration. Assume that the PTP protocol works on top of the layer 2 management protocol ((Rapid) Spanning Tree, Resilient Packet Ring, EAPS, etc.). In a ring topology the layer 2 management protocol spans a spanning tree by enabling and disabling the ring ports. The PTP protocol is able to send and receive PTP messages on enabled ports and cannot send or receive PTP messages on disabled ports.

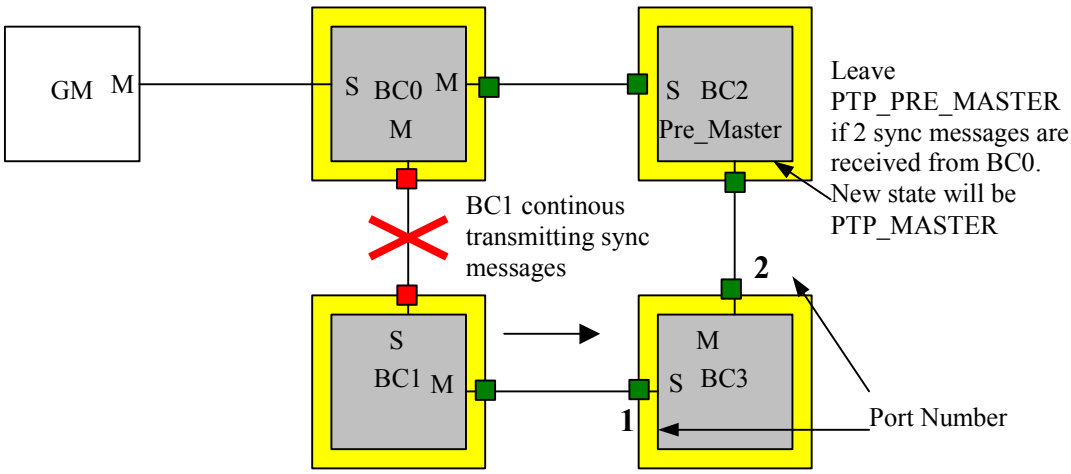


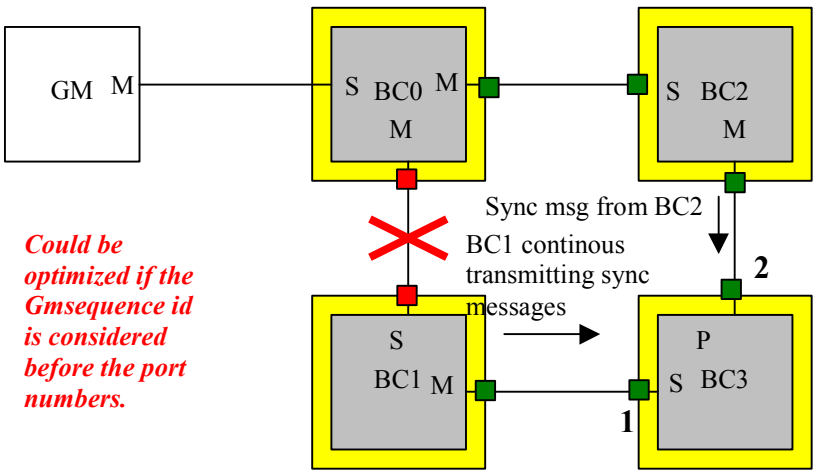
The figure below assumes that the connection between BC0 and BC1 is cut.



The layer 2 management protocol detects this fiber cut in less than 1 second (Rapid Spanning Tree) and reconfigures the ring network as shown in the figure below. So far the PTP protocol has not done any changes.

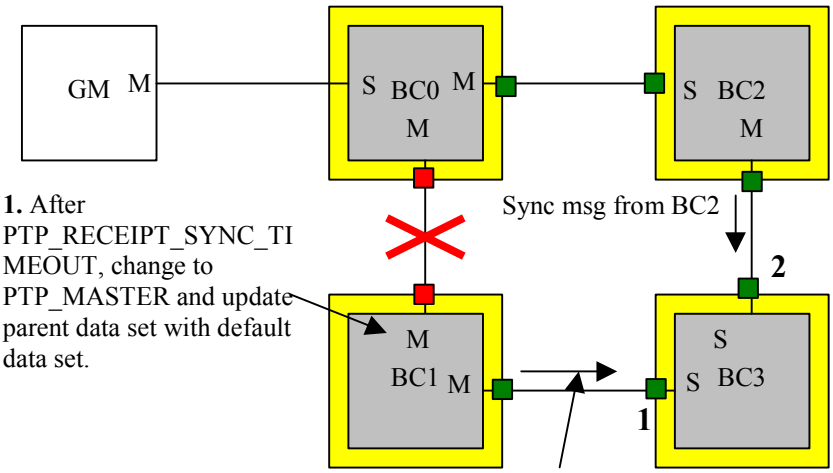

Ring topology: Ethernet port *active* via Rapid Spanning Tree
GM M
S BC0 M
M
M BC2
S
Context of layer2
Management protocols. E.g.
Spanning Tree, Rapid
Spanning Tree, Resilient
Packet Ring, etc.
disabled Ethernet port
Context and state of PTP
S
BC1 M
M
S BC3


GM M
S BC0 M
M
S BC2
Pre_Master
Leave
PTP_PRE_MASTER
if 2 sync messages are
received from BC0.
New state will be
PTP_MASTER
BC1 continous
transmitting sync
messages
2
S
BC1 M
M
S BC3
1
Port Number



Could be optimized if the Gmsequence id is considered before the port numbers.

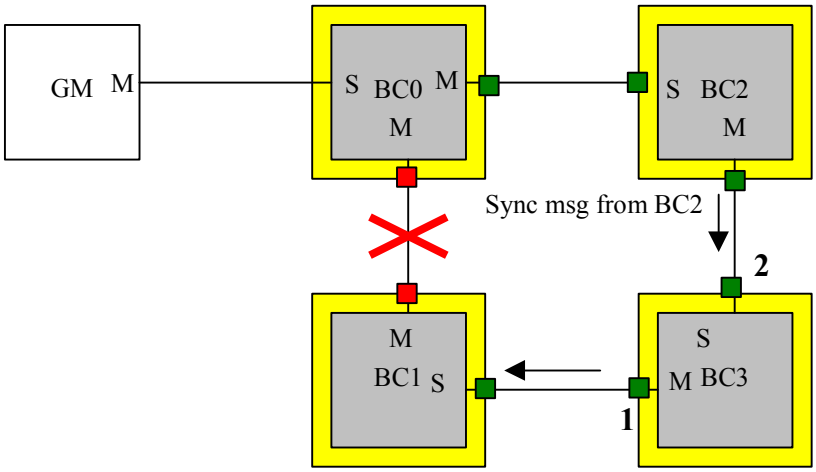
BC 3 compares sync from BC2 and BC1. Since port number from sync BC1 is 1 and smaller than 2 the data set comparision algorithm set port 2 of BC3 to **PTP_PASSIVE.** (Both sync messages have the same GM)



1. After PTP_RECEIPT_SYNC_TIMEOUT, change to PTP_MASTER and update parent data set with default data set.

3. BC 3 compares sync from BC2 and BC1. Since GMUUID and therefore the properties are different port 2 is set to PTP_SLAVE and BC3 to PTP_PRE_MASTER . After that it changes to PTP_MASTER.

2. Sends sync messages with new parent data set



This order causes sub-optimum (in terms of responsiveness) results in reconfiguring ring Networks. The 1588 algorithm converges but not as fast in this case as it would if the following suggestion is adopted. It occurs only if the steps removed fields are equal, to enter in the Data set comparison algorithm, part 3 Figure 19. If it is the case, the algorithm verifies first the PortnumberReceivingA with the PortNumberReceivingB. I think we should first check the GrandmastersequenceId of A and the GrandmastersequenceId of B to know which is the newer Sync Message. Otherwise we might trust the parent data set of the wrong BC for PTP_SYNC_RECEIPT_TIMEOUT seconds. If we first compare the GM_IDs the ring will be reconfigured faster.

Response:
The fundamental issue raised here is the trade-off between responsiveness to changes in network topology or clock properties and stability of the 1588 configuration with respect to transient conditions such as a missed packet. As written, the standard favors stability with respect to transients over responsiveness, with the threshold being 10 missed Sync messages. Bear in mind that a topology reconfiguration involves much more than simply picking new paths and possibly new grandmaster clocks. A change in topology also requires recalibration of the path length if the highest synchronization accuracy is to be achieved. This is one of the areas that needs operational experience to see if the current threshold is too conservative.

A second issue posed by the questioner is the interaction of two independent but not orthogonal protocols that can change the logical or physical topology of the network, in this case IEEE-1588 and any of the spanning tree protocols commonly used in Ethernet environments e.g. 802.1w. There are several points here:

- It is quite likely for many applications using IEEE-1588 that the underlying network does not use something like 802.1w. This still leaves the question (above) of whether the current threshold (10) in 1588 is too conservative.
- The proposed reversal of checking sequence Ids before port number equality may result in incorrect topology using 1588 in the absence of 802.1w. The logic represented by Figures 18, 19 and 20 are only invoked when the selection of master clock is based on topology.

- The goals of the topology portions of IEEE-1588 and a protocol like 802.1w are not the same. 802.1w seeks to configure the network as a minimum spanning tree. By contrast, if the subdomain contains multiple stratum 1 or 2, or preferred clocks, IEEE-1588 seeks to first partition (logically for purposes of 1588) the network into regions each served by a grandmaster clock. IEEE 1588 does this partitioning such that each region forms a minimum spanning tree and to the extent possible the depth of the trees in the regions are similar.

The tradeoff between stability and responsiveness is an important issue and should be readdressed at the next revision based on actual experience.

QUESTION 2: Loss of FollowUp messages (submitted March 19, 2003)
Consider the case where we receive qualified SYNC messages and the PTP_ASSIST flag is set. The receipt of qualified SYNC messages implies that we don't change the state (PTP_SLAVE) – if no better master appears–, but we do not synchronize the clock to this master. If we loose all associated Follow-Up messages we do not synchronize the local clock and this is never detected. You might think that this is academic but consider the case where you have created two threads. The first listens on the event port and the second on the general port. If the general port thread dies we cannot receive Follow-Up messages. Ok what are the solutions ?
We could observe the threads. But the Follow-Up messages could be lost by some other errors.
We could re-trigger the SYNC_RECEIPT_TIMEOUT if

- PTP_ASSIT = TRUE: we have received a Follow-Up message
- PTP_ASSIT = FALSE: we have received a SYNC message (as it already is)

Response:
It is true that the standard specifies that when PTP_ASSIST is TRUE that the local clock should be adjusted only on the basis of the timing information in Follow_Up messages, which as the question notes can be missed for a variety of reasons. Irrespective of the reason for failure to receive a Follow_Up message for each Sync message, such failure can only be detected by the receiving clock. Clause 7.5.18 notes that "..any condition that if persistent will:..Prevent the correct execution of the protocol defined by this standard, … or Produce or continue faulty behavior of the local clock" is a FAULT_DETECTED event. Inability to synchronize due to failure to receive Follow_Up messages from the master at intervals sufficient to keep the local clock synchronized consistent with the specifications of the local clock falls within this definition of a FAULT.

The standard, 7.5.18, does not specify how this detection is to be implemented nor of the options implementers have for clearing such a fault. Note that if a FAULT_DETECTED event is declared that by 7.3.1 the clock will have to be initialized before resuming normal action.

A less draconian option is to declare this a SYNCHRONIZATION_FAULT, which by 7.3.1 only results in the slave node going to the PTP_UNCALIBRATED state which only requires a re-assessment of the best master clock before returning to the PTP_SLAVE state.

Even if the slave node wished to effect a change of master clock the only method consistent with the normal operation of the protocol is specified by the standard in clause 7.5.14

involving SYNC_RECEIPT_TIMEOUT_EVENT. The changing of the master clock is an ensemble action bought about by all nodes executing the protocol. The only other recourse to changing the master clock accepted by all nodes is the use of various management messages to cause resets or other actions. The use of management messages for this purpose has not been addressed by the standard. The intent of such messages is to form the basis for products designed to monitor the operation of 1588 systems and to provide user directed corrective action.

It is quite true that Sync messages could be received but not the corresponding Follow_Up messages when the PTP_ASSIST flag bit is TRUE, and vice versa. However the decision on which, of possibly several clocks for which Sync (and possibly Follow_Up messages) have been received, should be the master clock can only be decided on the basis of information contained in the Sync messages. Since this is a distributed algorithm all nodes must act on the same information and with the same procedure, which is the basis for clause 7.6. It is therefore not possible to base the SYNC_RECEIPT_TIMEOUT mechanism on Follow_Up messages since they lack the necessary information to determine whether the sending clock should remain as master.

QUESTION 3: Processing of Delay_Req messages (submitted March 19, 2003)
The averaging interval shall be PTP_DELAY_REQ_INTERVAL x PTP_SYNC_INTERVAL_TIMEOUT seconds. That is 30 x PTP_SYNC_INTERVAL_TIMEOUT. But we generate a random number between 2 and PTP_DELAY_REQ_INTERVAL (30). On average we receive a Delay_Req message every PTP_DELAY_REQ_INTERVAL/2 * PTP_SYNC_INTERVAL_TIMEOUT. That is 15 x PTP_SYNC_INTERVAL_TIMEOUT.

RESPONSE:
The issue here is whether the node can process Delay_Req message fast enough to keep up with the receipt of such messages. This refers to material in clause 7.11.3. The relevant numbers from this clause are (using the values from 7.9):
Averaging interval = $(30-2)x2^{sync\_interval}$ seconds
Mandated average processing rate = $(18-2)/(2^{sync\_interval}) = 16/(2^{sync\_interval})$ messages per second
Average time between Delay_Req issued by a single clock = $(30+2)x(2^{sync\_interval})/2= 16x(2^{sync\_interval})$ seconds
Average rate of Delay_Req from a single clock = $1/(16x\ (2^{sync\_interval}))$ messages per second
Number of slave clocks on a communication path within a sub-domain to produce the average rate =
$16/(2^{sync_interval})$messages per second ÷ $1/(16x(2^{sync_interval}))$messages per second = 16x16 = 256 clocks
The implication is that for up to 256 clocks on a communication path within a sub-domain and conformant to the standard, Delay_Req messages will be processed correctly on average.

QUESTION 4: Management (submitted March 19, 2003)
We would like to implement the precision time protocol on an Ethernet switch. The de facto management protocol is SNMP.

What do you think about the management via SNMP ?

RESPONSE:
SNMP is certainly a significant tool in the IP world. As such it makes sense to consider it in conjunction with IEEE-1588. The initial version of IEEE-1588 provides only PTP management messages for roughly the same purposes. SNMP should be considered at next revision. Discussion in the user community prior to revision is appropriate. Some considerations are:

- IEEE-1588 allows for implementation on non-IP networks Care will be needed to ensure that any requirement or specification of SNMP does not restrict the choice of network beyond restrictions already inherent in IEEE-1588. For example it might be recommended as an alternative in addition to management messages in IP networks.
- SNMP at present requires knowledge of the IP address of each node to be interrogated. This implies an administered system and the use of non-SNMP discovery protocols. IEEE-1588 management messages provide for discovery of IEEE-1588 nodes and for multicast as well as directed query and update.
- Some management message functions, such as changing sync_interval or initialization, are best accomplished using multicast rather than point-to-point. This would be difficult using SNMP, at least in its present form.

QUESTION 5: MIBs (submitted March 19, 2003)
Has or does anyone define an IEEE1588 MIB for SNMP management purposes? If not we could define an IEEE1588 MIB for the annex and submit it as RFC.

RESPONSE:
To our knowledge no one has defined an IEEE1588 MIB. As noted this is an appropriate topic for discussion in the user community and during first revision of the standard. Some thought needs to be given to the relationship of the IEEE standardization process to the IETF. Our understanding is that the IETF deals only with Ethernet and the IP community and does not consider other network protocols. This was one of the reasons for choosing the IEEE standards process. However if the IETF makes recommendations for IEEE-1588 on Ethernet or IP networks then it surely is possible to incorporate these into the next revision. At a minimum this could be done by adding to the existing Annex D the requirement to incorporate an IETF specified MIB in Ethernet/IP implementations.

QUESTION 6: (submitted March 19, 2003)
Does there exist a PTP management tool, which uses either the PTP management messages or SNMP?

RESPONSE:
Not to our knowledge although this does not seem like a difficult task. What has been done, by Agilent in their prototypes, is to incorporate IEEE-1451.2 TEDS (roughly analogous to a MIB), which allows easy generation of web-based management of IEEE-1588 functionality

via normal commercial browsers. These prototypes were done before the completion of IEEE-1588 and do not provide access to all of the functionality defined by the management messages, but the extension would be trivial. Browsers have the same problem as SNMP in that they are point-to-point and it will therefore be difficult to produce system wide multicast updates.

QUESTION 7: (Submitted 8 July 2003)

In the IEEE 1588 standard publication, there appears to be a typographical error in Figure 16.

In the decision block 'Ebest better than Erbest determined by length, the source of dataset update information appears to be backward in the resultant recommended states. If Ebest is better than Erbest (the 'yes' path), the resultant state is PTP_PASSIVE, and the source of dataset update information is Erbest. Shouldn't the source of the dataset update information be Ebest in this case?

RESPONSE:
Reference clause 7.6.3.

The figure is correct as it stands. However in the case of decisions P2 and M3 the contents of the parenthesis, Ebest or Erbest, is not relevant to the update process since only the port state is updated. This is also true for decision P1. The port state is updated to Passive for P2 and P1 and to Master for M3. See clause 7.6.5 tables 17, 19 and 20.

Decisions P2 and M3 occur only for boundary clocks. Note that figure 16 refers to operations on port R.

Decision P2 is reached on port R when Ebest is better by path length and the state or R is to be set to passive as a means of breaking cyclic 1588 communication paths. The port on the boundary clock that is in the slave state (choice S1) updates the parent data set while R only updates the port state on R.

Decision M3 is reached on port R when some other port (on which Ebest was seen) will be the slave (choice S1). This slave port will be the port that updates the parent data set while port R only changes the port state on R.

It probably would have been clearer just to leave the parenthesis blank for these choices. This will be recommended to the committee at next revision.

QUESTION 8: (Submitted 30 July 2003)

I am trying to understand the 'port' concept of 1588 with respect to Ethernet implementations and boundary clocks. As an example, a PC with 2 Ethernet cards (equipped with clocks and 1588 hardware assistance) could be considered a boundary clock. One Ethernet card is on one network, and the other is on a different network.

In such a scenario, one of these cards could be the master clock of it's network, if it has the best clock. The other might be a slave, if another clock on it's network is better. The BMC algorithm is supposed to figure this out.

My difficulty is with the way most networks operate, and with the way systems use port numbers. Both these Ethernet cards, and in fact an ATM card and a GigE card can all be attached to port 319 and 320, and the system will receive packets from all of them. There is then a difficulty in distinguishing the 1588 'ports' from each other, and a difficulty in assigning the states appropriately.

My questions:
1-In an Ethernet (or other network) implementation, are we supposed to equate 1588 ports to network ports? If so, what other ports should be used besides 319 and 320? If I have a boundary clock with 3 network cards, I need 3 pairs of ports.

2-If we are supposed to use just the 2 1588 ports, what is the recommended mechanism to distinguish between the 1588 ports? Do I use their MAC address?

3-The way the BMC algorithm is laid out, I need to look at the data on all 1588 ports and decide the best (Ebest). If I have a clock (or clock reference) on every Ethernet card, why do I need to consider data from 1 port when deciding the state of another? Is this intended for the scenario where the Ethernet cards are using a timing source from another card (like a separate GPS receiver)?

RESPONSE:
Background: IEEE 1588 communication topology is illustrated in Figures 2 & 3 of clause 6.2. A 1588 system is made up of multiple PTP communication paths, for example paths A, B, C and D in these figures. Within each path all 1588 activities act independently of the other paths. This means that nodes such as 13, 14, and 15 do not pass any 1588 messages directly from one communication path to another. In Ethernet technology nodes such as 13, 14, and 15 could be routers or switches configured to block 1588 messages. If for example node 14 was a repeater then paths B and C would merge into a single path.

Within a single path, e.g. A, there will be a single clock in the master state. This may be one of the ordinary clocks or the ‘port’ of node 13 communicating with path A. The function of a boundary clock, e.g. nodes 13, 14, and 15 is to synchronize the times in each of the communication paths it touches. This is discussed in clause 7.1 and 7.2. Typically one port of a boundary clock will be in the slave state and synchronize to a master clock on the communication path associated with that port. The other ports on the boundary clock will typically be in the master state and other clocks on the associated communication paths will synchronize to them. Within a boundary clock of n ports the boundary clock implementation must decide which port can see the ‘best clock’ and therefore be the slave port. It is possible that the internal clock of boundary clock is the best clock visible. This is done by comparing the various datasets associated with each port of the boundary clock. The BMC algorithm, clause 7.6, sorts all of this out.

Answer to questions 1&2:

See clause 3.26 for the definition of a port. It is a logical address in the relevant communication technology. Every 1588 clock access to a communication path has two such logical ports, a general and an event port. In Ethernet technology, see Annex D, these ports have been assigned the numbers 319 and 320 by the IANA. In a boundary clock with n associated communication paths, there will therefore be n Ethernet logical ports with number 319, and n with 320. (In your example of a PC with multiple Ethernet cards and assuming that you are implementing boundary clock technology in the PC then each card would have sockets opened with port numbers 319 and 320.) Each physical port (made up of two logical ports the event and general ports) also has a UUID, see clause 6.2.4.1. This UUID, specifically the port_id_field, is used internally by a boundary clock in executing the BMC algorithm to distinguish among the various logical ports on the boundary clock. Clause D.2 defines how to generate the first two fields of the UUID in an Ethernet implementation. The assignment of the port_id_field is implementation dependent.

Answer to question 3: The general answer above covers most of the question. A boundary clock, just like an ordinary clock, must decide which of all of the clocks visible is the 'best'. In the case of a boundary clock this means that it must examine the clocks visible on each port (i.e. on each accessed communication path) and its own internal clock to determine the 'best' clock. If there is an external source of time, for example a GPS receiver, there are two implementation choices. First the GPS receiver and its associated clock may be implemented as the 'local clock' of Figure 6. The second choice is to have the GPS receiver and associated clock be visible to the boundary clock only via one of its ports, in which case it appears like any other clock on the associated communication path. Presumably, being a GPS clock, the BMC executing in all of the clocks, including the one associated with the port of the boundary clock, will determine that the GPS clock is the 'best clock visible' and it will be the master clock in the communication path, and likely the grandmaster clock of the entire system if there are no other similar GPS clocks on other paths.

QUESTION 9: (question submitted May 3, 2004)
One issue I see for developers, system integrators, and customers/users is determining the accuracy of the clocks in a given system. How do you measure this in a standard way? A simple mechanism might be to add 2 management messages (not much thought put into the names):

PTP_MM_TIME_SNAPSHOT - The receiving port captures the time, as measured by the local clock, at which the message timestamp passed the inbound clock timestamp point (corrected for latency). The clock will then issue a PTP_MM_TIMESTAMP in response.

PTP_MM_TIMESTAMP - Contains the receipt timestamp of a previously issued PTP_MM_TIME_SNAPSHOT.

A client application can issue the PTP_MM_TIME_SNAPSHOT using the default clock UUID and receive the time value of each local clock when the message was received. The only differences would be in the one-way-delays between the client app and each target clock. With no hops and a localized system, that delay should be very small, especially if the local clocks are close together on a machine. The Delay Request mechanism could be used to improve this, but is not allowed to be acted on by a slave.

Does this seem reasonable? It does provide a client app with a standard way of verifying a system of 1588 clocks without using scopes, etc., at the expense of some accuracy. For some users, sub microsecond times are not required. Rather 10 - 100 microsecond accuracy is what they want, and need to verify. I believe any accuracy caused by transmission delay would be negligible in that case.

Or is there a better way that I have not considered?

RESPONSE:
The question of an independent verification of synchronization performance is difficult.

There are actually two issues. The first is whether the system and all components are correctly executing the protocol. The second is given that the protocol is being executed correctly are the clocks actually synchronized to the specified accuracy.

It is not possible to give a positive answer with high confidence to either of these questions within the protocol itself. It may be possible to give a negative answer to one or both under certain conditions. For example if multiple masters in the same domain and communication path persist in sending Sync messages beyond several PTP_FOREIGN_MASTER_TIME_WINDOW periods then the mechanism for selecting a master clock is not working.

The questioner's proposal would give a rough measure of performance to the 10 us or so depending on topology. A more fundamental problem is that the proposed method is just a variation on what happens with a Sync message, and the Followup message, in the normal operation of IEEE 1588. The slave uses the timestamps to estimate the offset and servo the slave clock to zero. To have any value an independent message, one not involved in the control loop, would have to be used. The problem is that it would require clocks to timestamp a message other than the Sync message from the current master, which is a pretty big addition particularly if the timestamping is done in hardware, as it will be in most high accuracy implementations.

There are two management messages designed to help in determining system performance, among other things. The message PTP_MM_CURRENT_DATA_SET delivers the current values seen at a slave of the offset from the master and the computed one-way delay. The message PTP_PARENT_DATA_SET provides the slaves view of the drift and variance of the master. This is not quite the same information but can serve as a basis for a 'reasonableness' test.

At this point there does not appear to be any scheme that has been sufficiently proven to warrant inclusion in the standard.

QUESTION 10: (question submitted May 3, 2004)
The transition from PTP_LISTENING to PTP_MASTER is not clear.

RESPONSE:
This transition, and others from the PTP_LISTENING state are defined in 7.3.1 Figure 9. Note that in this figure there are three 'multi-state' state boxes all of which contain the state

PTP_LISTENING and which are designated by heavy lines per 4.3. The event causing a transition from PTP_LISTENING to PTP_MASTER is clearly indicated as the SYNC_RECEIPT_TIMEOUT_EXPIRES event. Transitions from PTP_LISTENING to the states PTP_DISABLED, PTP_FAULTY, PTP_PRE_MASTER, PTP_PASSIVE, and PTP_UNCALIBRATED are also possible and are clearly defined by events in Figure 9.

QUESTION 11: (question submitted May 3, 2004)
Modifiable parameters (Preferred Master, Sync Interval) are not specified as non-volatile. The Sync Interval is implied to be required as non-volatile based on the fact that the setting does not take effect until 'clock initialization or power up / reset'. I believe all it should be specified that all settable parameters be non-volatile.

RESPONSE:
Non-volatile storage of modifiable data is discussed in clauses 7.4.1 and 7.12.4. It appears that these clauses are sufficiently specific to cover the question; however this may be worth reviewing at future revision. Note the response to a similar question on 7.12.4 in the interpretations documents.

QUESTION 12: (question submitted May 3, 2004)
The value 0 parameter of the Clock Initialization management message does not indicate that the specification defaults should be stored in non-volatile memory. Either a new value should be defined which provides for this, or the existing value 0 should indicate that the defaults are made permanent.

RESPONSE:
It appears that the specification in 7.12.4 is clear. A messageParameter value of zero indicates that the result of the PTP_MM_INITIALIZE_CLOCK message is to cause the recipient clock to update any modifiable data to the initialization values. It is true that the standard does not say how to accomplish this but this is normal standards practice unless the means is somehow critical to the operation of the specification. Note the response to a similar question on 7.12.4 in the interpretations documents.

QUESTION 13: (question submitted May 3, 2004)
The specification, clause 7.12.4, has divided the clock initialization parameter into public and vendor/product specific ranges. The public range consists of 2 values, while the vendor/product specific range consists of 65,534 values. Both of the public values are already defined, leaving no room for expansion. Further, the vendor/product specific ranges in the 1588 spec are of very limited use. If not too late, the public range should be greatly expanded.

RESPONSE:
This is a good comment and should be considered at the next revision. Until then it is recommended that implementation specific values be greater than 512.

QUESTION 14: (question submitted May 3, 2004)
Reference 8.2.1 and 8.2.2. If a clock sees a 'newer' version (higher value) should it fault? Certainly, it cannot guarantee compatibility. I did not see any reference to a use of the version fields. If we are going to count on them in the future, we may want to consider documenting

and enforcing compliance to whatever use is intended. And if not, why are they sent in every message. Similar questions arise if new Flags fields or new values for MsgType and Control fields appear.

RESPONSE
This is an important point unaddressed by the standard and should be considered at the next revision. The intent was to provide a mechanism for dealing with evolution of the standard and compatibility, to the extent possible, of mixed systems, hence their inclusion in messages.

It is recommended that until the standard is revised to be explicit on this issue that implementations of this version of the standard ignore the version numbers.

Note that defining incompatible versions as a fault precludes backward compatibility for future revisions. If a future revision cannot be made such that backward interoperability is assured then the only logical recourse is to specify that beyond some revision systems must not use components complying with an earlier revision.

New MsgType or Control field values would represent a major change in the operation of the standard. If this occurs and cannot be done in a backward compatible way (which seems likely) then nothing that we can say about the current standard is likely to help. Flags fields are more likely to represent additional features rather than new functions so ignoring new values is more likely to be successful.

QUESTION 15: (question submitted May 3, 2004)
The specification has fixed the 'timeout multiplier' at 10, clause 7.9 table 24. Thus, for the default sync interval setting of 2 seconds, a loss of master will require at least 20 seconds before any slave detects the loss of that master. The x10 multiplier seems rather long. A reduction in this value could be considered, possibly via a management message.

RESPONSE:
There are several things to consider here. The specification in 7.9 is in terms of a multiple, 10, of the sync interval rather than an absolute time interval. The tradeoff is between smaller values of this multiple and increased potential thrashing of the selection of best master clock due to dropped or corrupted Sync messages. In a compact topology this multiplier might be reduced but in larger topologies this could make things much worse since changes in best master clock take at least one sync interval for each boundary clock along a path. The usual argument for making this value smaller or decreasing the Sync interval is to respond more quickly to changes in topology, for example due to a cut communication link, to preserve the accuracy of the system time base. However this particular problem can be addressed in other ways, for example by more stable oscillators that provide better holdover characteristics. Note that thrashing will be worse if this value is not the same in all components. However there may be circumstances when this timeout value is too large for some operational conditions such as recovery from a network fault. Future revision committees may wish to provide for this, for example via a management message that forces the PTP_SYNC_RECEIPT_TIMEOUT event.

QUESTION 16: (question submitted May 3, 2004)
I did not see in the specification where the target of a Management 'Request' determines what the target UUID fields of a response should be. As a client sending the request, it is obviously up to that application to decide whether to go 'point to point' or 'broadcast' (using defaults). As the 'server', it only seems to make sense to load the source UUID as the target UUID. However, the spec leaves it open (or so it seems to me). Is the target device really allowed to do either? I do not see a value in a 'broadcast' response. If 'point to point' response is the intent, we should say it and conformance check for it.

I believe that in all cases the 'target' should return the complete Source UUID as the Target UUID (including the target port). Any device 'snooping' can do so regardless of the value in these fields, so the 'broadcast' is not needed. By returning the source, the request integrity is maintained.

For those messages that do not require the target port field, the specification should declare what the value is (i.e. zero).

Specifying a value of zero for the Target port field when not required and including an echo of the source port should be added to all 'response' type management messages.

RESPONSE:
The clauses of interest are 8.6.1.3 & 8.6.1.4. The questioner raises a reasonable concern and poses a reasonable solution. The standard as it stands is specific even though there is no guidance on which choice to take for 'response' style management messages. Likewise there are no protocol faults introduced by the available choices. The advantage of limiting the choice for 'response' messages is reduction in network traffic. Note that this issue has no effect on the propagation of management messages of any sort through boundary clocks. This behavior is specified in clause 6.2.2.1. A similar argument can be made for the targetCommunicationTechnology field.

This issue should be revisited during the next revision of the standard. Until that time it is recommended that for the management messages:

- PTP_MM_CLOCK_IDENTITY
- PTP_MM_DEFAULT_DATASET
- PTP_MM_CURRENT_DATASET
- PTP_MM_PARENT_DATASET
- PTP_MM_PORT_DATASET
- PTP_MM_GLOBAL_TIME_DATASET, and
- PTP_MM_FOREIGN_DATASET

the targetCommunicationTechnology, targetUuid and targetPortId fields be set to the sourceCommunicationTechnology, sourceUuid and sourcePortId fields respectively of the management message requesting the information in this management message. For management messages not listed above for which the targetUuid and targetCommunicationTechnology fields are assigned the default values, the value of the targetPortId should be set to zero.

QUESTION 17: (question submitted May 3, 2004)
Section 7.5.20.3 shows the reference point of the external timing signal as the start of the SFD. In the appendix D.1.1 it seems that the start of the 1 byte after the SFD is the point of reference. Are these different reference points intended (if "yes": why? / is it a mistake)?

RESPONSE:
7.5.20.3 defines a timing point on the EXTERNAL TIMING SIGNAL. This is an OPTIONAL out of band signal for use in special applications. The external timing signal is a continuous signal with a distinguishing mark as defined in 7.5.20.3. This mark is NOT a start of frame mark indicating the beginning of user data in a packet but rather markes the second boundary of the local timebase.

D.1.1 defines the message timestamp point on a Sync or Delay_Req message in an Ethernet environment.

QUESTION 18: (question submitted May 3, 2004)
The standard uses the expression “drift” and “drift rate”. Is the meaning the same? (The standard contains no explicit definition of the terms.)

RESPONSE:
The standard needs to be more precise and should be corrected in the next revision. Clause 7.4.4.10 gives the clearest indication of intent. 7.4.4.10 defines the observed_drift as a drift rate of an observed clock with respect to a local clock measured in nanoseconds per second. The intent in all uses of either “drift” or “drift rate” in the standard is to define how rapidly two clocks diverge with time as noted in 7.4.4.10.

QUESTION 19: (question submitted May 3, 2004)
The SOF count is incorrect in D.1.3.2.

RESPONSE:
True. The mistake starts in the row with SOF currently indicated as 148 and should clearly be 146. All subsequent rows are in error by 2 as well.

QUESTION 20: (question submitted May 3, 2004)
There is no mechanism for acknowledgement of receipt and processing of a management message other than by observing a clock's behaviour or issuing a query of the data sets to see if they have in fact changed. A similar situation exists on a burst request in a Delay_Req message.

RESPONSE:
The questioner is correct. As noted in the response to a question on 7.12.4 in the interpretations documents a means of viewing pending but not instantiated updates would be useful. Clause 7.4.6.10 specifies how a clock declares whether it is capable of requesting and supplying burst behavior. This is viewable from the port data set. Future revision committees may wish to consider whether lumping both requesting and providing in the same parameter is sufficient and whether this is a static or dynamic property but for the moment the specification is clear.

QUESTION 21: (question submitted May 3, 2004)
There are a number of requirements, for example setting preferred master 8.6.1.10.6 and 8.6.1.10.7 that make no sense when received by a slave only or management only node.

RESPONSE:
True but there are no errors introduced by this. It is recommended that future revision committees review the entire standard and explicitly specify which requirements may be made optional or ignored for less than fully conformant clocks, that is currently management only or slave only clocks.

QUESTION 22: (question submitted May 3, 2004)
Currently initializable is a static property. It would seem that this ought to be dynamic to allow clocks to permit or deny initialization depending of their state or perhaps external state.

RESPONSE:
This is worth considering in future revisions but for the moment the specification is consistent. The sort of circumstance that might make a dynamic version of initializable useful would be for a node to deny such service when other applications are being reprogrammed. This would suggest a management message to enable, disable and view this property. This would, among other things, require modification of the second option in clause 7.4.1.